2010

SOAC: A conceptual model for managing service-oriented authorization

Haiyang Sun
Macquarie University

Weiliang Zhao
University of Wollongong, wzhao@uow.edu.au

Jian Yang
General Research Institute for Non Ferrous Metals,Ministry of Science & Technology, China,Macquarie University

2010 IEEE International Conference on Services Computing

SOAC: A Conceptual Model for Managing Service-Oriented Authorization

Haiyang Sun, Weiliang Zhao, and Jian Yang
Department of Computing, Macquarie University, Sydney, NSW2109, Australia
{hsun,wzhao,jian}@ics.mq.edu.au

Abstract

A web service can be composed of multiple component web services in a loosely-coupled environment. Traditional Role Based Access Control (RBAC) is inadequate for the authorization management of composite services since the administration of the component web services has not been taken into consideration. In this paper, we propose a novel conceptual model, named as Service Oriented Authorization Control (SOAC) to facilitate the administration and management for both service consumers and component web services. A set of administrative functions are also provided for managing the elements of SOAC. This research will be the first step towards managing service-oriented authorization.

Keywords*: Authorization, Conceptual Model, Web Services, Role Based Access Control.*

1 Introduction

Web service technologies provide a technical foundation for seamlessly composing individual component web services into a cohesive one [1]. However, how to manage the access on the composite web service becomes a challenge in loosely-coupled environment.

Let us look at an example. Tom & Brothers is a vehicle parts dealer which can provide vehicle engines, engine accessories, etc, for both military and civil use. An *Order Service* is set up in Tom & Brothers including five operations: (1) *Order Engine*, (2) *Order Engine Accessory*, (3) *Payment*, (4) *Payment Verification* and (5) *Logistics* (See Fig. 1). Note, the *Logistics* operation is not available to the military customers since they organize the parts shipment by themselves. When receiving a part order from a customer, the Tom & Brothers will order the parts from various other parts suppliers. As soon as the payment has been verified, the goods will be transported to the customer.

We can observe the following challenges of managing authorization for the composite web service-*Order Service* in Tom & Brothers:

1. **Complicated Coordination on Authorization Constraints**: Each web service in Tom & Brothers or other part suppliers bears specific authorization constraints to restrict the access on its operations. The *Order Service* is a composite web service and its operations are supported by multiple component web services provided by other part suppliers. It is not enough to enforce the authorization constraints of the *Order Service* without considering the characteristics of these component services. For example, in Fig. 1, the operation (1)-*Order Engine* is supported by the component web services A, B and C from other organizations that can provide engine. Hence, without properly handling the authorization constraints of the component web services, the Tom & Brothers can not ensure if the authorization to access the specific *Order Service*'s operations can be supported. For instance, if the authorization to access the *Order Engine* operation has been assigned to a specific service consumer, but Tom & Brothers fails to obtain the authorization from the other component web services, then the operation can not be enacted and the unnecessary disclosure of *Order Engine* operation is occurred. This is the result of lack of coordination on the authorization constraints in *Order Service* and its supporting component web services. Therefore, granting the access on the *Order Service* to a service consumer needs to consider not only the service consumers but also the component services. We should not only understand "`who should do what`?", but also need to know "`who should do what under what conditions`".
2. **Dynamicity of Service Environment**: Web services are autonomous and interact with each

978-0-7695-4126-6/10 $26.00 © 2010 IEEE
DOI 10.1109/SCC.2010.13

other in a loosely-coupled environment. Many web services are composed of component services in a highly dynamic manner. For example, if a component service changes its authorization constraints from asking Tom & Brothers for professional engineer certificate to requiring sales representative qualification, then all the service operations in Tom & Brothers that can be supported by the component web service need to update their identifications on the component service's authorization constraints. Moreover, there are huge amounts of web services that can provide the same or similar operations. For example, in Fig. 1, component services A, B, and N can all support the same type of engine accessories to Tom & Brothers. Hence, if the changes occur frequently and/or happen in thousands of web services, then an efficient way to administrate these changes is needed. At this stage, the dynamicity of web Service impedes the *efficiency* of service-oriented authorization management.

3. **Conflict of Interest**:

   There are different kinds of conflicts of interest which are crucial in the authorization of composite web services. For example, when there is a conflict of interest between a specific service consumer and a specific component service, the *Order Service* in Tom & Brothers should not be authorized to the service consumer when this component service is needed in the composite service, e.g., USA military customers have a conflict of interest with a component web service from a Chinese part supplier. Actually, the conflicts of interest associated with characteristics of component services have not been touched in existing research about web services authorization.

Therefore, an effective management on service-oriented authorization by coordinating the constraints in different web services is needed. Role Based Access Control (RBAC) [2] is a widely accepted approach to restrict system access to authorized users. In RBAC, users acquire permissions through their roles rather than they are assigned permissions directly. This greatly reduces the administrative overhead associated with individual users and permissions.

However, web services technologies facilitate the integration of the loosely-coupled distributed applications. There may be a large amount of component web services which are used as resources to support the composite web service's operations. The quantity of service consumers can also be large. As illustrated in Fig. 1, the system of the composite web service (*Order Service*) needs to deal with not only a large number of users (service consumers) but also great amounts of resources (component services). Moreover, the characteristics of these resources are different from that of objects of RBAC in close systems. These resources do not belong to the concerned system, but the system can reach their requirements and provided operations. Hence, traditional RBAC is not suitable for the service-oriented authorization management since it has not taken the *Resource* into account. Note the *Resource* particulary can not be fully controlled in the context of service environment.

All existing role-based models in web service paradigm have not brought the administration of resource into the picture. Actually, the quantity of resources can be very large and they can be prone-to-change, which should be considered in web service authorization. In research work [4, 6, 7], roles are assigned to service consumers for service authorization. However all these researches have not put the resource into the picture or they simply employ an unrealistic assumption that there is a *global* coordination on internal authorization policies of each autonomous web services to enforce the access control in service composition.

In this paper, we propose an innovative conceptual model for authorization of web services, named as *Service Oriented Authorization Control (SOAC)*. The conceptual model will handle the web service authorization by dealing with not only a large number of service consumers, but also huge amounts of resources. Authorization constraints of the component web service and its supported composite web service are integrated together for making authorization decision to a service consumer. Furthermore, administrative functions are also presented to enforce the web service authorization from a system perspective.

The rest of paper is organized as follows. Section 2 describes the conceptual model with the major features and detailed specifications. Section 3 discusses the administrative functions. Section 4 overviews some related work. Concluding remarks and discussion of future work are presented in Section 5.

2 Conceptual Model of Service Oriented Authorization Control

We propose a novel conceptual model, named as *Service Oriented Authorization Control (SOAC)* for managing the access on the composite web service (See Fig. 2). Based on the proposed conceptual model, the elements of service authorization and their internal relationships can be systematically described. We

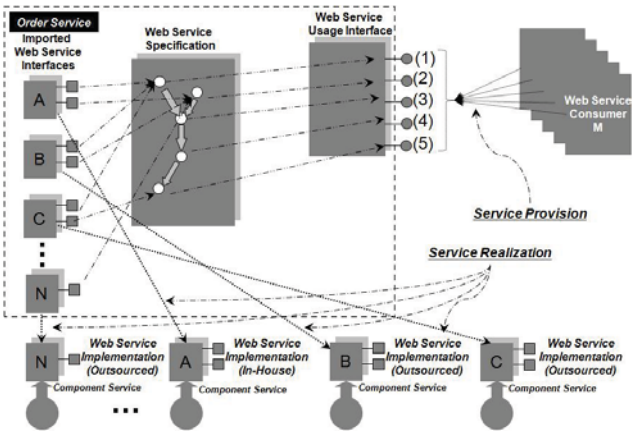

Figure 1. *Order Service* **in Tom & Brothers**

develop the SOAC conceptual model by using the notation of **Entity-Relationship** (**E-R**) **Diagram**. In Fig. 2, the rectangles represent the elements and the diamonds illustrate the relationships.

2.1 Features of Service Oriented Authorization Control

In SOAC, the authorization management concerns two parts as *service provision* part (See grey rectangle in Fig. 2) and *service realization* part (See white rectangle in Fig. 2). In *service provision* part, the concept of **Role** is employed for handling with the administration and management of profiles of service consumers. A role will be assigned to a specific service consumer to gain certain permissions for accessing the operations of the composite web service, if the service consumer can satisfy the authorization constraints of the composite web service. For example, a web service consumer can be identified as USA military customer to obtain permissions to access all operations of the *Order Service* in Tom & Brothers except the *Logistics* operation, if it bears the certification from Pentagon. In a summary, role in SOAC is used to encapsulate a group of web service consumers that can satisfy the common authorization constraints of the composite web service.

Following the same philosophy of roles for service consumer's profiles, the conceptual model introduces **Resource Type** to deal with the large number of the component services (resources). In *service realization* part, the design of resource type is based on the characteristics and authorization constraints of the component services. In SOAC, resource type is used to represent a group of resources that bear common authorization constraints and can provide same support to the composite web service. From the motivating example, the *Order Engine* operation in Tom & Brothers requires the support from the resource type named as `Engine Supplier` that contains the component web services A, B, and C which can provide engine (See Fig. 1). Obviously, one resource type can include multiple resources, and vise versa.

The reasons to introduce the resource type in SOAC are: (1) the authorization to a service consumer to access the composite web service's operations relies on not only the profile of the service consumer but also the characteristics and authorization constraints of the component web services; (2) resource type is more constant compared with individual resources. It is easy for the composite web service to map its operations with resource type than individual resource which are dynamic in loosely-coupled environment. For example, if one resource changes its authorization constraints to be in another resource type, then the composite web service only need to transfer the resource from this resource type to another one. It is not necessary to change the relationship between the resource types and the supported composite web service's operations. Without resource type, the composite web service has to update all relationships between this resource and the involved composite service's operations. If the change is frequent and the affected resources and operations are countless, then the workload to specify the change individually will be huge. Therefore, resource type is particularly necessary in service oriented environment where the resources are thousands and very dynamic, and the relationships between the resources and their supported operations are loosely-coupled.

2.2 Specification of Service Oriented Authorization Control

In this section, we introduce the specification of SOAC from three perspectives, (1) *service provision* part, (2) *service realization* part, and (3) integration of these two parts.

2.2.1 Specification of *Service Provision* Part

A typical authorization constraint of the composite web service requires that only the service consumer that can fulfill the composite service's constraints, e.g., bearing specific credential, can be assigned with specific roles to gain the permissions to access the operations of the composite web service (See *Constraint* enacted between the elements of Role (**R**) and Service Consumer (**SC**) in Fig. 2). In Fig. 2, we define service consumer

as the element that requires to access the composite web service's operations. To avoid confusion between the accessed composite web service's operation with the element of *Operation* used in traditional RBAC, we introduce a new element-function to represent the composite web service's operations. The function can be represented by one composite service's operation or several operations. Since there is only one type of permission for service consumer to act on the composite web service's function in service environment - "execute the function", we ignore the permission in SOAC.

Since service consumer is prone to change, and its quantity is vary large, directly specifying the assignment of function to individual service consumer needs tedious administration effort. In SOAC conceptual model, we follow the philosophy of RBAC to have the more constant concept-role to encapsulate the service consumers which can fulfill the common authorization constraints of the composite web service. A role will be assigned to each service consumer based on its characteristics (typically a credential that service consumer submits to the composite web service). Each role binds with a group of functions that can be accessed. The roles guarantee that the composite web service's functions can only be accessed by the qualified service consumers as specific roles. In a summary, what we are concerned in SOAC is `"who should act on what as what kind of role"`. Here below, we present the formal definition of *service provision* part of SOAC.

Definition 1 *The service provision part of SOAC includes:*

- ***SC***, ***R*** ,*and* ***F*** *are elements representing Service Consumer, Role, and Function.*
- **SCA** $\subseteq$ ***SC*** $\times$ ***R***, *a many-to-many relation to map service consumer to role assignment. Formally,* $\forall s^c \in SC$, $\forall r \in R$, $(s^c, r) \in SCA \Rightarrow s^c.credential = r.credential$, *where the credential that the service consumer submits is consistent with the credential that the role requires.*
- ***assigned_sc:(r:R)*** $\rightarrow 2^{SC}$, *the mapping of role r onto a set of service consumers. Formally,* $assigned\_sc(r) = \{s^c \in SC | (s^c, r) \in SCA\}$.
- **PFA** $\subseteq \boldsymbol{F} \times \boldsymbol{R}$, *a many-to-many relation to map function to role assignment.*
- ***assigned_fun:(r:R)*** $\rightarrow 2^{F}$, *the mapping of role r onto a set of functions. Formally,* $assigned\_fun(r) = \{f \in F \mid (f, r) \in PFA\}$.

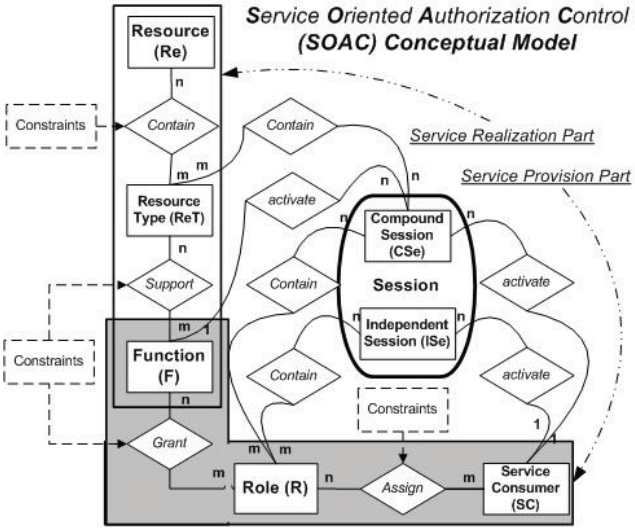

Figure 2. Service Oriented Authorization Control (SOAC) Conceptual Model

2.2.2 Specification of *Service Realization* Part

When a composite web service receives the authorization request from a service consumer, at first it needs to confirm if the realization of the required functions can be supported by specific type of the component web service, i.e., resources. Due to the feature of **Dynamicity** of the component web service, it is unrealistic for the composite service to specify the relationships between the resources and the supported composite web service's function individually, if the resources and the functions are numerous. Hence, the concept resource type is introduced in SOAC. The resource type is a set of resources defined by the composite service after identifying the characteristics and authorization constraints of the resources (See Fig.2). The composite web service can bear multiple resource types that include many resources. The resource therefore can be used to support the function if the function is mapped with the resource type that contains the corresponding resource. The resource type can contain the resources if the authorization constraints of the resources can be satisfied by the resource type and the support that the resource can provide is what the resource type needs (See *Constraint* between the elements of Resource Type (**ReT**) and Resource (**Re**) in Fig. 2). Here below, we present the formal definition of *service realization* part of SOAC.

Definition 2 *The service realization part of SOAC includes:*

- ***F***, ***ReT***, *and* ***Re*** *are elements representing Function, Resource Type, and Resource.*
- **SFA** $\subseteq$ ***F*** $\times$ ***ReT***, *a many-to-many relation to map function to resource type.*
- ***assigned_ret:(****ret****:ReT)*** $\rightarrow 2^F$, *the mapping of resource type ret onto a set of functions. Formally, assigned_ret(ret)=*$\{f \in F | (f, ret) \in SFA\}$.
- **RTA** $\subseteq$***Re***$\times$***ReT***, *a many-to-many relation to map resource to resource type. Formally,* $\forall re \in Re$, $\forall ret \in ReT$, $(re, ret) \in RTA \Rightarrow re.constraint = ret.constraint$, *where the constraint that restricts the access on the resource is consistent with the constraint that the resource type can fulfill.*
- ***assigned_re:(****ret****:ReT)*** $\rightarrow 2^{Re}$, *the mapping of resource type ret onto a set of resources. Formally, assigned_re(ret)=*$\{re \in Re | (re, ret) \in RTA\}$.

2.2.3 Specification of Integration of Two Parts in SOAC

In previous two sub sections, we have introduced the elements and their relationships in *service provision* part and *service realization* part of SOAC. These two parts are logically correlated with each other and are required to be integrated together at runtime for service-oriented authorization.

In Fig. 2, the mappings between the elements of Role (**R**), Function (**F**) and Recourse Type (**ReT**) logically integrate the *service provision* part and *service realization* part together. Hence, the access on the composite web service can be assigned to service consumer if the constraints of the composite web service and its resources can both be satisfied, i.e., the service consumer in *service provision* part can be assigned with a specific role to access the functions if the corresponding functions can be supported by the relative resource type which includes specific resources in *service realization* part. We summarize the authorization rules in Lemma 1:

Lemma 1 *Authorization to access the composite web service's functions can be granted to service consumer in service composition if:*
(1) The service consumer can satisfy the authorization constraints of the composite web service.
(2) The resource type can satisfy the authorization constraints of the resources (the component web services).
(3) The functions that the role needs to access can be logically supported by the resource type.

Here below we introduce each condition of Lemma 1 in detail. In **condition (1)**, it stresses the constraints enacted between the elements of Service Consumer (**SC**) and Role (**R**) shown in Fig. 2: "only service consumer that can satisfy the authorization constraints of the composite service can be assigned as specific role". On the other hand, **Condition (2)** is used to restrict the relationships between Resource Type (**ReT**) and Resource (**Re**). It states that the resource type can only include the resource whose authorization constraints can be satisfied by the resource type. Furthermore, **condition (3)** emphasizes that the function that the resource type can support should be what the role needs to access (See *Constraint* among the relationships between the elements of Role (**R**), Function (**F**) and Resource Type (**ReT**) in Fig. 2). For instance, if the function that a service consumer needs to access represents *Order Engine* operation in Tom & Brothers, then the resource type deployed in SOAC should be `Engine Suppler` that can logically support the function, i.e. the resource type includes the specific resources that can provide engine to Tom & Brothers.

In Fig. 2, another element **Session** is introduced in SOAC to integrate the two parts of SOAC together at runtime. There are two types of sessions in SOAC, *Independent session* (**ISe**) and *Compound session* (**CSe**). When service consumer starts to send message to the composite web service by using its functions with specific assigned roles, the service consumer and its activated roles are included in one independent session. When the message arrives at the composite web service, the composite service thereby needs to use the supporting resources for processing the message. We believe that the resource type is activated when the message is passed from the composite web service to the specific resources that belong to the resource type. At that time, a compound session is created by combining the resource type and supported function with the corresponding service consumer and activated role. Note, the resource type and the supported function can not be included in the independent session, since a composite web service can not use the resource type without receiving the message from the service consumer. Here below we present the formal definition of **Session** at integration of two parts in SOAC.

Definition 3 *The integration of two parts in SOAC includes:*

- ***Session*** *includes two types, Independent Session (****ISe****) and Compound Session (****CSe****), where:*
  - ***Independent Session (ISe)*** *is used by service consumer* s^c *to map the set of activated roles* $\{r_1..r_j\}$, $j \geq 1$.
  - ***Compound Session (CSe)*** *is used by a group of service consumer and function <*

$s^c, f >$, *where service consumer* s^c *requires to access the functions of the composite web service f, to map a set of activated roles and resource types* $\{< r_1, ret_1 >..< r_j, ret_k >\}$, *where:*

- $r_1..r_j$, $j \geq 1$, *is a subset of roles assigned to and activated by the specific service consumer* s^c.
- $ret_1..ret_k$, $k \geq 1$ *is a subset of resource type assigned and activated to support the specific function f.*

- ***Service Consumer Independent Session:*** $SCSi{:}(s^c{:}\mathbf{SC}) \rightarrow 2^{ISe}$, *the mapping of service consumer* s^c *onto a set of independent sessions ISe.*
- ***Service Consumer Compound Session:*** $SCSc : (s^c : SC) \rightarrow 2^{CSe}$, *the mapping of service consumer* s^c *onto a set of compound sessions CSe.*
- ***Role Independent Session: RSi:*** $(se_i{:}\mathbf{ISe}) \rightarrow 2^R$, *the mapping of independent session* se_i *onto a set of roles.*
- ***Role Compound Session****:* $RSc : (se_c : CSe) \rightarrow 2^R$, *the mapping of compound session* se_c *onto a set of roles.*
- ***Function Session****:* $FS(f{:}F) \rightarrow 2^{CSe}$, *the mapping of function f onto a set of compound session CSe.*
- ***Resource Type Session****:* $RTS(se_c{:}CSe) \rightarrow 2^{ReT}$, *the mapping of compound session* se_c *onto a set of resource types.*

3 Administrative Functions of SOAC

In this section, we present the administrative functions of SOAC to outline the semantics of various administrative operations. They are required for maintaining the SOAC conceptual model components, e.g., the element sets and relations. We also take the **Session** into consideration to manage the activation of the specific elements. Three categories of administrative operations are presented: (1) Service Provision Administrative Operation ($\mathcal{SP}-\mathcal{AO}$); (2) Service Realization Administrative Operation ($\mathcal{SR}-\mathcal{AO}$); and (3) Session Administration Operation ($\mathcal{SE}-\mathcal{AO}$). The notation used to formalize the administrative operations is basically a subset of the **Z** notation. The representation schema in the formal specification of the administrative operations is:

$Schema{-}Name(Declaration) \triangleleft Predicate; ...; Predicate \triangleright.$

Table 1. Service Provision Administrative Operation - $\mathcal{SP}-\mathcal{AO}$

$\mathcal{SP}-\mathcal{AO}_{ele}$
(1) `AddServiceConsumer(serviceConsumer:NAME)`
(2) `DeleteServiceConsumer(serviceConsumer:NAME)`
(3) `AddRole(role:NAME)`
(4) `DeleteRole(role:NAME)`
(5) `AddFunction(funtion:NAME)`
(6) `DeleteFunction(funtion:NAME)`
$\mathcal{SP}-\mathcal{AO}_{rela}$
(7) `AssignServiceConsumer(serviceConsumer,role:NAME)`
(8) `ResignServiceConsumer(serviceConsumer,role:NAME)`
(9) `GrantFunctionAccess(function,role:NAME)`
(10) `RevokeFunctionAccess(function,role:NAME)`

All the data types and operations used in the formal specification are defined in SOAC definition in previous Section. *NAME* is an abstract data type used to represent the identifiers of the elements in SOAC.

3.1 Service Provision Administrative Operation- $\mathcal{SP}-\mathcal{AO}$

There are three elements in *service provision* part of SOAC-Service Consumer (**SC**), Role (**R**), and Function (**F**), and two relationships among the elements-*Assign* and *Grant* (See grey rectangle in Fig. 2). Hence, $\mathcal{SP}-\mathcal{AO}$ are separated into two aspects $\mathcal{SP}-\mathcal{AO}_{ele}$ and $\mathcal{SP}-\mathcal{AO}_{rela}$ to manage the elements and relationships respectively.

The operations (1)∼(6) (See Table. 1) are $\mathcal{SP}-\mathcal{AO}_{ele}$ used to add and delete elements in *service provision* part. Due to space limit, we only take the operation (4) as example. In operation (4), an existing role is deleted. If the role is activated by the specific service consumer in independent session or compound session, then the associated session should be deleted firstly. Furthermore, the mapping functions and relations associated with the deleted role also need to update to erase the effect of this role. Then the role can be deleted from the set **R**.

```
(4) DeleteRole(role:NAME)◁
  role∈R;
  [∀ se_i ∈ISe • role ∈ RSi(se_i)⇒
    DeleteIndependentSession(se_i)];
  [∀ se_c ∈CSe • role ∈ RSc(se_c)⇒
    DeleteCompoundSession(se_c)];
  SCA'=SCA\{sc:SC • sc↦ role};
  assigned_sc'=assigned_sc\
    {role↦assigned_sc(role)} ;
```

PFA'=PFA\{f:F • f ↦ role};
assigned_fun'=assigned_fun\
{role↦assigned_fun(role)};
R'=R\{role};▷

The operations (7)∼(10) (See Table. 1) are $\mathcal{SP}-\mathcal{AO}_{rela}$ used to add and delete relationships in *service provision* part. For example, Operation (9) are used to create the relationships between the elements of Role (**R**) and Function (**F**).

(9) GrantFunctionAccess(function,role:NAME)◁
function∈F; role∈R;
PFA'=PFA∪{function↦role};
assigned_fun'=assigned_fun\
{role↦assigned_fun(role)}∪
{role↦(assigned_fun(role)∪{function})};▷

3.2 Service Realization Administrative Operation- $\mathcal{SR}-\mathcal{AO}$

There are also three elements included in $\mathcal{SR}-\mathcal{AO}$-Function (**F**), and Resource Type (**ReT**), and Resource (**Re**). However, the element of function has already mentioned in previous sub section. We only introduce the two elements left in $\mathcal{SR}-\mathcal{AO}_{ele}$. Moreover, two relationships-*Support* and *Contain* are included in $\mathcal{SR}-\mathcal{AO}_{rela}$ (See white rectangle in Fig. 2).

Operations (11)∼(14) (See Table. 2) are used to create and delete elements of resource type and resource. They follow the same rules enacted in $\mathcal{SP}-\mathcal{AO}_{ele}$. Operations (15)∼(18) (See Table. 2) aim to create and delete the relationships-*Support* and *Contain* in *service realization* part by using the same rules in $\mathcal{SP}-\mathcal{AO}_{rela}$.

3.3 Session Administrative Operation- $\mathcal{SE}-\mathcal{AO}$

$\mathcal{SE}-\mathcal{AO}$ is used to maintain the element of **Session** and associated relationships, which reflects the activation of the corresponding elements. Four commands are developed in $\mathcal{SE}-\mathcal{AO}_{ele}$ based on the creation and deletion of two types of **Session** (See Operations (19)∼(22) in Table. 3). The operations (23)-(34) are $\mathcal{SE}-\mathcal{AO}_{rela}$ used to maintain (add and delete) the six relationships associated with the element of session (See Fig. 2). Due to space limit, we do not list the twelve operations of $\mathcal{SE}-\mathcal{AO}_{rela}$ in Table 3. Let us take the operation (21) as an example to illustrate the $\mathcal{SE}-\mathcal{AO}$. Operation (21) is used to create a compound session. Firstly, an *ars* is created

Table 2. Service Realization Administrative Operation - $\mathcal{SR}-\mathcal{AO}$

$\mathcal{SR}-\mathcal{AO}_{ele}$
(11) AddResourceType(resourceType:NAME))
(12) DeleteResourceType(resourceType:NAME)
(13) AddResource(resource:NAME)
(14) DeleteResource(resource:NAME)
$\mathcal{SR}-\mathcal{AO}_{rela}$
(15) SupportFunction(function,resourceType:NAME)
(16) AbandonFunction(function,resourceType:NAME)
(17) ContainResource(resource,resourceType:NAME)
(18) ExcludeResource(resource,resourceType:NAME)

which represents the activated roles and resource types included in this new compound session. The mapping functions *SCSc(serviceConsumer)* and *FS(function)* are also updated to reflect the new compound session. Finally, the *RSc*(se_c) and *RTS*(se_c) are updated to point to the activated roles and resource types from the compound session se_c.

(21) CreateCompoundSession
(serviceConsumer,function:NAME;
ars:2^{NAME};se_c:CSe)◁
serviceConsumer∈SC; function∈F;
ars⊆{(role:R, resourceType:Ret)|
(serviceConsumer↦role)∈SCA,
(function↦resourceType)∈SFA};
se_c∉CSe; CSe'=CSe∪{se_c};
SCSc'=SCSc\
{serviceConsumer↦SCSe(serviceConsumer)}∪
{serviceConsumer↦
(SCSe(serviceConsumer)∪{se_c})};
FS'=FS\{function↦FS(function)}∪
{function↦(FS(function)∪{se_c})};
RSc=RSc'∪{se_c↦ars}; RTS=RTS'∪{se_c↦ars};▷

4 Related Work

In [2, 3], a basic Role-based Access Control (RBAC) model is presented. The security policy in RBAC does not directly grant permissions to users but to appropriate roles. However, traditional RBAC model is only suitable for authorization management within individual organization. In RBAC, the resource that is required to support the function is not considered, since it is assumed as a constant concept, which quantity is small, and can be fixed in advance and less

changed. However, in loosely-coupled environment, resource that is needed to support the function spreads across-organizational boundary and is composed in highly dynamic fashion. Hence, the dynamicity of resource in service environment makes the authorization management complicated. Research has been done in service composition security by enhancing RBAC. We shall look into some representative works in the area .

In [4, 5], the authors provide an enforcement and verification approach to guarantee that a service choreography can be successfully implemented between a set of web services (service consumer and composite web service), based on their authorization constraints. However, the paper did not mention how to manage the access control after the authorization constraints of the composite web service are satisfied. The authorization constraints of the supporting resources are totally ignored.

The authors in [6, 7] propose an approach to correlate the local role issued by the individual component service with the global role generated from the composite services. However, the "role" as a concept used by specific service to manage the authorization is part of internal security policy within each web service and can not be identified by the other services. Hence, the mapping of the global role with the local role is not realistic. In SOAC, we introduce the resource type (**ReT**) as the composite web service's acknowledgment on the public authorization constraints of the resources.

Although plenty of existing enhanced RBAC mechanisms and approaches have been presented which focus on managing access control in service composition, they are still insufficient in: (1) missing the administration of the resources in service-oriented authorization; (2) ignoring the dynamicity of service environment where the composite web service is composed of resources based on-demand; (3) hard coding the roles issued from resources and composite service together.

In this paper, we present an innovative conceptual model, SOAC, to manage the service-oriented authorization. The merits of SOAC lie in: (1) the coordination of the authorization constraints of composite web services and component web services; and (2) the introduction of the more constant concepts of role and resource type to represent the concepts of service consumer and resource which are dynamic and volatile.

5 Conclusion and Future Work

This research work provides an extension of classical RBAC approach with the capability to address the authorization issues of the composite web service, brought in by the large number of dynamic service consumers and component web services. A novel conceptual model has been developed for managing the service-oriented authorization in the loosely-coupled environment of web services. Beyond the existing approaches for web services authorization, our approach considers the authorization constraints of the component web services explicitly. Three categories of administrative functions of SOAC are also proposed.

In the future, we will develop mechanisms to elaborate the causes and solutions regarding to conflicts of interest in web service authorization. Detailed and formalized authorization constraints in SOAC and a context-aware extension will also be studied.

Table 3. Session Administrative Operation - $\mathcal{SE} - \mathcal{AO}$

$\mathcal{SE} - \mathcal{AO}_{ele}$
(19) `CreateIndependentSession` `(serviceConsumer:NAME;ars:`2^{NAME}`;`se_i`:NAME)`
(20) `DeleteIndependentSession` `(serviceConsumer,`se_i`:NAME)`
(21) `CreateCompoundSession` `(serviceConsumer,function:NAME;` `ars:`2^{NAME}`;`se_c`:NAME)`
(22) `DeleteCompoundSession` `(serviceConsumer,function,`se_c`:NAME)`

References

[1] M. Papazoglou, D. Georgakopoulos.: Service-Oriented Computing. Communications of the ACM **46**(10) (2003) 25–28.

[2] RS. Sandhu, E. Coyne, H. Feinstein, C. Youman.: Role-based Access Control Models. IEEE Computer **29**(2) (1996) 38–47.

[3] D. Ferraiolo, J. Cugini, R. Kuhn.: Role Based Access Control: Features and Motivations. In: Proceedings of ACSAC. (1995).

[4] M. Mecella, M. Ouzzani, F. Paci, and E. Bertino.: Access Control Enforcement for Conversation-based Web Service. in Proceedings of the International World Wide Web Conference, (2006), 257–266.

[5] F. Paci, M. Ouzzani, and M. Mecella.: Verification of Access Control Requirements In Web Servies Choreography. in Proceedings of SCC, (2008), 5–12.

[6] R. Wonohoesodo, and Z. Tari.: A Role Based Access Control for Web Services. in Proceedings of SCC, (2004), 49–56.

[7] J. Fischer, and R. Majumdar.: A Theorey of Role Composition. in Proceedings of ICWS, (2008), 49–56.